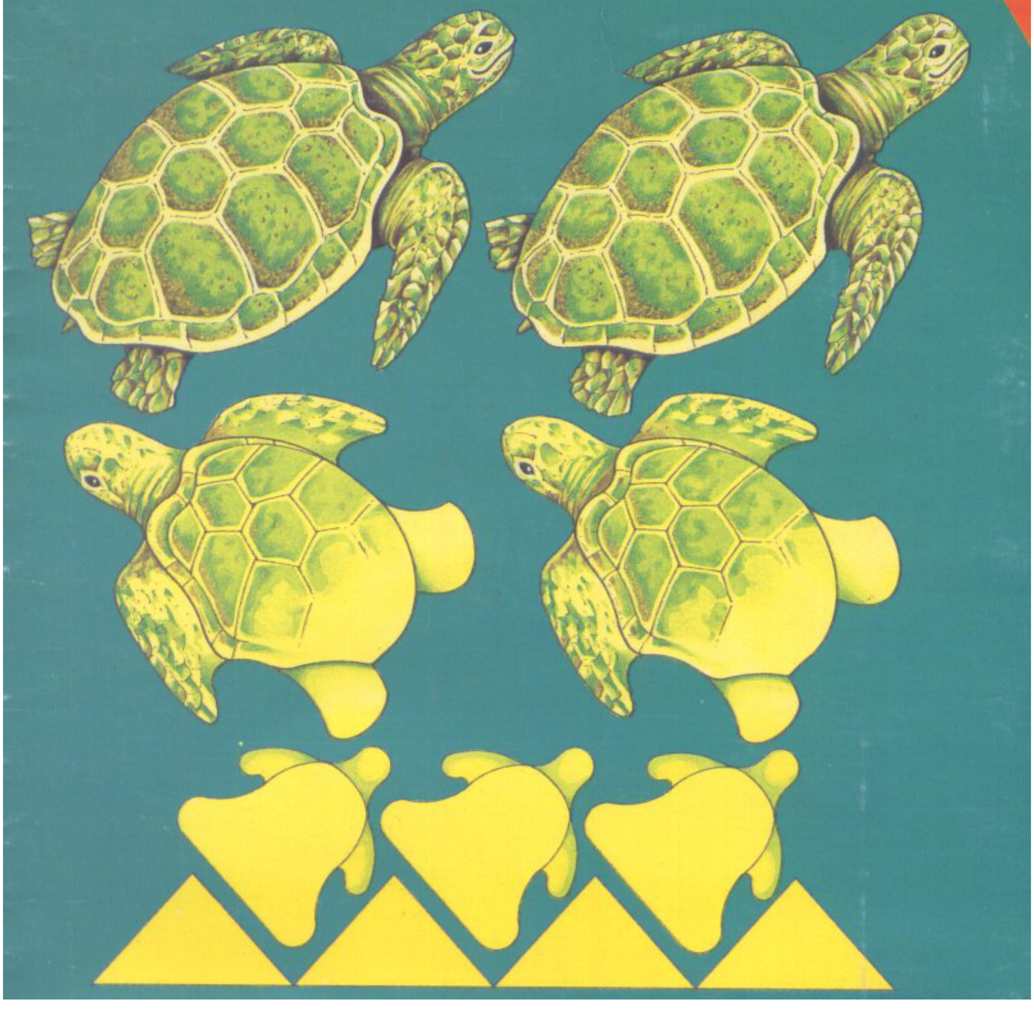
sinclair

ZX Spectrum

Logo 2



Sinclair Logo 2
Programming Reference Manual

Please Note:

This manual was Scanned, OCR-ed and PDF by
Stephen Parry-Thomas 29 March 2004.
For ZX-Spectrum Users and to preserve the manual.

Sinclair Logo 2
Programming Reference Manual

by Ellen Sparer
and the editorial staff of SOLI/LCSI

This edition first published in 1984 by

Sinclair Research Ltd
25 Willis Road Cambridge CB1 2AQ England
ISBN 185016 018 X

Production supervised by Business Literature Services Ltd;
Typeset by Goodfellow & Egan Ltd;
French's Mill, French's Road, Cambridge CB4 3NP

Printed in England by Staples Printers St Albans Limited at The Priory Press.

Contents

Neither the authors nor the publishers can be held liable for any direct, indirect, incidental or consequential damages relating to the use of the LOGO program and documentation.

While every effort has been made to ensure the accuracy of the LOGO program and documentation, the authors and publishers cannot be held responsible for any error which may occur.

The authors and publishers reserve the right to change the LOGO manuals, glossary and software at any time and without notice.

Les Systems d'Ordinateur
Logo International
SOLIIntl
33 Rue de Poissy
75005 Paris
FRANCE

Logo Computer Systems Inc
LCSI Inc
9960 Cote De Lisse
Lachine Quebec
CANADA H8T 1 A1

Preface

This is a reference manual for experienced Logo users, rather than a guide for newcomers. If you are a newcomer, you should start by reading *Sinclair Logo 1 - Turtle Graphics*. If, however, you have used Logo before, this manual may be all you need to understand Sinclair Logo.

Sinclair Logo 1 gives full details of how to set up your Sinclair ZX Spectrum, how to load Logo, and how to use turtle graphics.

Sinclair Logo 2 gives concise descriptions of the Logo primitive procedures - or primitives - the in-built terms of the language, and each is illustrated with examples. We can show you the way with one or two examples; we hope that you will then make up some examples of your own in order to test the full power of Sinclair Logo.

The first chapter of this Manual is a general overview of Logo grammar; the rest define and explain the use of each of the primitives, grouped according to their interrelationships.

If you wish to find a particular definition or primitive, consult the index at the back of the Manual.

If you are looking for a primitive for a particular task within a program, check the table of contents, where you will find the primitives listed by category.

Chapter 1
A Summary of Logo grammar

INTRODUCTION

As with all languages, Logo has a *grammar* - certain basic rules for writing and combining the building blocks of the language. In this section we will describe how to use this grammar, so that Logo understands what you want it to do.

PROCEDURES

One of the powerful aspects of Logo is its ability to work with procedures (building blocks). There are two kinds of procedures: those which Logo 'knows' as they are present every time you load your Sinclair Logo (they are called *primitive procedures*) and those which you define yourself.

For example, if you type:

```
HIDETURTLE
```

the turtle disappears from the screen. Logo knows how to execute this action without being told. HIDETURTLE is a primitive procedure.
However, you can teach Logo new procedures:

```
TO WELCOME
PRINT [WELCOME TO SINCLAIR
 LOGO]
END
WELCOME defined
```

In this example, WELCOME is a procedure which prints the list [WELCOME TO SINCLAIR LOGO] when Logo is asked to WELCOME.

Note: the **first** line of a procedure you define yourself is called the TITLE LINE. It always begins with **TO** followed by the name of the procedure.

The last line always contains the word **END** by itself.

There is an important difference between defining a. procedure and asking Logo to execute it. We are defining a procedure where we tell Logo how TO WELCOME. We execute it by typing WELCOME when the ? prompt appears on the screen.

If you enter WELCOME after the ? prompt, Logo will execute the procedure WELCOME and reply with:

```
WELCOME TO SINCLAIR LOGO
```

However, we can also call a procedure indirectly. In the procedure WELCOME we have called the primitive procedure PRINT. You can write procedures using previously defined procedures, a facility which makes Logo a particularly powerful language.

```
TO LONGWELCOME
WELCOME
PRINT [I THINK WE'LL HAVE FUN]
PRINT "GOODBYE"
END
LONGWELCOME defined
```

Here WELCOME is a *subprocedure* of LONGWELCOME. And LONGWELCOME is the *superprocedure* of WELCOME.

```
LONGWELCOME
WELCOME TO SINCLAIR LOGO
I THINK WE'LL HAVE FUN
GOODBYE
```

If you type a word that has not been defined as a procedure you will get a Logo message; for example, type:

```
JEAN
```

```
I don't know how to JEAN
```

OBLECTS

Logo *objects* are words or lists used as inputs to or outputs from procedures.

A *word* is a series of alphabetic or numeric characters. A word is contained within two delimiters (see next section). Each character in a word is said to be an *element* of that word.

A quote mark at the beginning of the word enables Logo to distinguish words from procedure names. There is also a word with no characters; called the *empty word*; it is written with a single quote mark.

```
PRINT "R2B2
R2B2
PRINT "WELCOME
WELCOME
```

In this example both "R2B2 and "WELCOME are Logo words.

Numbers are also words in Logo, but you can write them without the quote mark.

```
PRINT 25
25
```

The words concerned with logic, TRUE and FALSE, may also be written without the quote mark.

```
PRINT "TRUE
TRUE
PRINT TRUE
TRUE
PRINT "FALSE
FALSE
PRINT FALSE
FALSE
```

A *list* consists of a series of Logo objects; ie, words or other lists. A list is usually enclosed by brackets. The individual elements that comprise the list are separated by blank spaces.

An empty *list* is written as [ ].

[CAT DOG MOUSE HOUSE] is a list containing four elements

[[CAT DOG] [HOUSE MOUSE]] is a list containing two elements, each of two elements.

[FORWARD 50 [2R 5B] BLUE] is a list containing four elements

Logo objects can be used as variable names. For example:

```
MAKE "WELCOME 38
```

In this example, WELCOME is not only the series of letters, W, E, L, C, 0, M, E, but it is now being used as a variable name and has the value 38.

DELIMITERS

A word is usually delimited by spaces at either end of it, which separate it from the rest of the line. However, there are some other *delimiters*:

```
[ ]( )  =  >  <  +  -  *  /
```

There is no need to type a space between a word and any of these characters although it is customary to do so for clarity.

For example:

```
1>2+(3+4)/5-6
1 > 2 + ( 3 + 4 ) / 5 - 6
```

are the same.

Note that care must be taken with the minus sign as only -6 is taken to be 'minus six'.

INPUTS

Some procedures need *inputs* to enable them to work. Inputs are Logo objects (words or lists); they may either be given explicitly, or appear as the outputs of other procedures.

```
PRINT [GOOD MORNING]
GOOD MORNING
```

Here the list [GOOD MORNING] is the input for the primitive procedure PRINT.

PRINT must be given an input as shown by:

```
PRINT
Not enough inputs to PRINT
```

When you define a procedure using TO, the title line must contain the word TO, the name of the procedure and the inputs for that procedure if any. Each input must be preceded by a colon.

```
TO MOOD :RESPONSE
PRINT CDEAR TURTLE, HOW ARE
   YOU?]
PRINT :RESPONSE
PRINT [PLEASURE TO TYPE TO
   YOU, GOODBYE]
END
```

When a procedure is called, the user must give the procedure name followed by the necessary number of Logo words that the procedure is expecting

```
MOOD "FINE
DEAR TURTLE, HOW ARE YOU?
FINE
PLEASURE TO TYPE TO YOU,
GOODBYE
```

QUOTES, COLONS, BRACKETS AND PARENTHESES

Logo interprets every undefined word as a re to run a procedure, unless you specifically indicate otherwise by preceding the Logo object with one of the following symbols:

The *quote mark* indicates to Logo that the chain of characters ending with a space which follows is a word.

The *colon* tells Logo that the chain of characters ending with a space which follows is the name of a Logo object and returns the contents of that Logo object.

Brackets tell Logo that the elements within them form a list.

Parentheses allow you to give more than two inputs to certain primitives.

```
PR (SENTENCE [I AM] [THE]
[GREATEST])
I AM THE GREATEST

PR (WORD "EN "THU "SIASM)
ENTHUSIASM
```

Parentheses also allow you to control arithmetic operations, (see below for further details).

```
PRINT 2 * 3 + 5
11

PRINT 2 * (3 + 5)
16
```

COMMANDS AND OPERATIONS

In Logo, a procedure can be either a command or an operation.

A **command** is a procedure which **does not output** a value.

An **operation** is a procedure which **does output** a value.

Consequently, any procedure that is an operation can act only as an input for another procedure.

For example:

```
TO ADD3 :A
A + 3
END

ADD3 4
You don't say what to do with 7 in
ADD3
```

results in a Logo message as the value produced does not form the input to a procedure. But:

```
TO ADDS :A
OUTPUT :A + 3
END
PR ADDS 4
7
```

is correct.

Note: the value from :A+3 is the input for OUTPUT. In its turn the value from ADDS 4 is the input for PR.

See what happens if you try to use a command as an input to another command:

```
PRINT FD 100
```

The turtle moves 100 steps; Logo returns a message:

```
FD does not output to PRINT
```

All the procedures you define yourself are either commands or operations.

VARIABLES

In Logo variables can be *named*.

Normally, variables are created using the procedure MAKE. For example:

```
MAKE "A 1
MAKE "B 2
```

And, the value of a variable can be obtained by using the : character before the name. (The procedure THING can be used instead of the : character, if desired.)

```
PRINT :A
```

will therefore search for the value attributed to the variable A and 1 will be outputted and used by the PRINT procedure.

Variables can hold a variety of data types and a user of Logo is not required to specify any prior details to Logo. For example:

`MAKE "A 1` would give A a single numeric value.

`MAKE "A [1 2 3]` would give A a list of three numeric values, in effect, forming a list of numbers.

`MAKE "A "APPLE` would give A a single word value.

`MAKE "A [APPLE PEAR GRAPES]` would give A a list of three words.

Data types can also be mixed. For example:

```
MAKE "A [2 APPLE 3 PEAR]
```

GLOBAL AND LOCAL VARIABLES

In Logo there are strict rules concerning global and local variables.

1 A variable created in Logo mode, by using MAKE, will be global. This means that variables named with MAKE exist both during and after execution of procedures.

2 A variable created within a procedure, by using MAKE, and not given as an input (ie, does not appear in the title line) to that procedure, will be global.

3 A variable declared as an input to a procedure, and not created previously, will be local to that procedure and any subprocedures. Once the procedure has completed all the instructions within it, the variable no longer has a value.

The following examples show these rules.

```
MAKE "A 22
PR 22
```

The variable A is global and will always have a value.

```
TO ONE
MAKE "B 24
TWO :B
END

TO TWO :B
PR :B
END
ONE
24
```

The variable B is created in the procedure ONE before it is used as an input to procedure TWO. It is therefore global.

However:

```
TO ONE
TWO 24
PR :B
END
TO TWO :B
END

ONE
B has no value in ONE
```

gives a Logo message as the variable B is local to the procedure TWO, and any subprocedures it might have, and thereby unavailable to the superprocedure ONE.

LOGO LINES

A Logo line can be much longer than a line on the screen. A Logo line may contain up to 242 characters - spaces included - and it ends when you press the ENTER key.

```
MAKE "PEOPLE [MEN WOMEN BOYS G !
  IRLS]
```

The ! indicates that the next screen line is a continuation of the same Logo line.

Here are some indications to help you to interpret a complex Logo line.

1 When you see a procedure name, be sure you know:

    Whether it is a command or an operation;
    How many inputs it should have.

2 The first procedure of a Logo line must always be a command.
3 An operation is written as input to another procedure.
4 Be sure to account for every input to a procedure.
5 When the inputs to a command have been accounted for, the next procedure must be another command.

Let's look at an example:

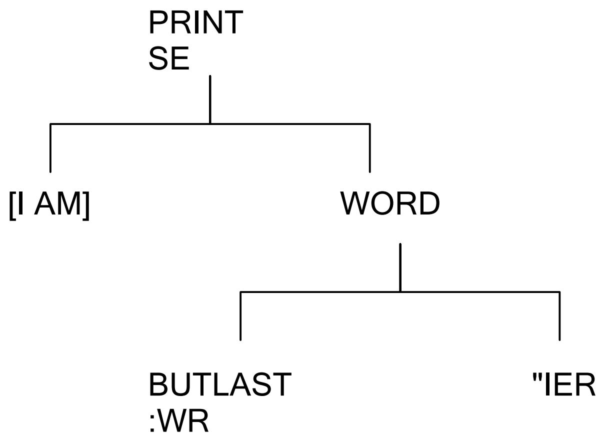
```
PRINT SE [I AM] WORD BUTLAST
  :WD "IER
```

Let's break down this Logo line.

PRINT is a command with a single input. This must be the output of SE, which is an operation with two inputs.

The first input to SE is the list [I AM]. The second is the output of the operation WORD. The latter is, once again, an operation with two inputs. The first is the operation BUTLAST, which has a single input :WD. The second input to WORD must therefore be "IER.

Since there are no more procedure names, and every input on the line has been accounted for, we have finished. The following diagram summarises what we have done.



So, for example, if the value of WD is HAPPY then the line would print I AM HAPPIER.

ARITHMETIC

Logo interprets numbers as words.
You do not need to put a quote mark before a number, although it makes no difference to Logo if you do.

```
MAKE "A "20
MAKE "B 20
PR :A
20
PR :B
20
```

The following priority is given to arithmetic operations:

Division,
Multiplication,
Subtraction,
Addition.

Division is executed before multiplication; both are executed before subtraction, which is executed before addition. This order can be changed by using parentheses; the contents of the parentheses are executed first.

```
PR 4 + 6/2
7
```

```
PR (4 + 6)/2
5
```

The use of parentheses is especially important when using operations such as RANDOM, SIN, TAN, etc.

```
PR RANDOM 2+3
```

is read by Logo as

```
PR RANDOM 5 (2 + 3)
```

and not:

```
PR (RANDOM2) + 3
```

Therefore, it is customary to write either:

```
PR RANDOM (2 + 3)
```

or better:

```
PR 3 + RANDOM 2
```

to avoid any confusion.

SCREENS, MODES AND PROMPTS

When using your Sinclair Logo, you have two possible screen options:

TEXTSCREEN: 22 lines are usable. The screen scrolls as your printing goes off the 'page'.

GRAPHIC SCREEN: the upper 22 lines are available for your drawing and the printing of text, the lower two lines are available for ordinary printing.

Certain modes will produce different effects when you type.

> **Logo mode** (direct mode): every instruction you give is interpreted and executed by Logo.
>
> **TO mode** (procedure writing): used for writing your own procedures.
>
> **Edit mode** (EDITOR): used for creating or modifying procedures.

Different prompts appear depending on the mode you are in.

> **Logo mode**: the prompt ? appears at the beginning of every Logo line.
> **TO mode**: the prompt > appears at the beginning of every screen line.
> **Edit mode**: the prompt ■ (flashing) appears at the top left of the screen.

RECURSION

Logo allows *recursion* and its use forms a powerful tool for the Logo programmer.

Recursion is said to occur when a *procedure calls itself*. Each call to the procedure by recursion forms a *recursive descent*, as the number of nested procedures increases. *Recursive ascent* occurs when control passes upwards through the nest of procedures; following **END** or stop commands.

Consider the following example:

```
TO DOUBLE :START
IF :START > 50 [STOP]
PR :START
DOUBLE :START * 2
END

DOUBLE 5
5
10
20
40
```

This example illustrates two important points.

1 A limiting test has to be included if the *recursive descent* is to be delimited. In the example the line IF :START > 50 [STOP] limits the program to three recursive calls.

2 Work can be performed during a recursive descent. In the example the line PR :START uses a variable whose value is being changed with each recursive call.

The next example shows how work can be performed recursive assent.

```
TO TREBLE :START
IF :START < 80 [TREBLE :START * 3 ]
PR :START
END
TREBLE 5
135
45
15
5
```

Note: Logo can be considered clever in the manner with which it handles a recursive ascent. Not only is the correct return position located, but the local variables for each level are available to the user.

EXITING FROM LOGO

If you wish to leave Logo without turning off your ZX Spectrum, type the command BYE. Sinclair Logo can be restarted by using RUN. The workspace is returned intact.

Chapter 2
The turtle

INTRODUCTION

The *turtle* is the name given to the little triangle which appears when you use the graphics screen. You can make it draw lines on your screen with its *pen* by asking it to move from one point to another - this is what *turtle graphics* is all about.

Every time you use a primitive related to the movement of the turtle, the *graphics screen* (including the turtle, unless instructed to remain hidden) will appear.

The graphics screen is devoted to the turtle's field, except for the two bottom lines where you and Logo exchange ideas. In Sinclair Logo the turtle field is normally 256 'turtle steps' across and 175 'turtle steps' high. The size of the turtle field can be changed by the user if desired (see SETSCRUNCH below).

The graphics screen disappears each time you change to text or edit mode.

The primitives that refer to the turtle will now be detailed:

BACK n
BK n

command

The turtle moves back n steps without changing its direction. (The turtle will move forward if n has a minus value.)

```
BK 20
```

BACKGROUND
BG

operation

Returns a number (0-7) representing the colour of the background.

The numbers of the colours are as follows:

0 Black
1 Blue
2 Red
3 Magenta
4 Green
5 Cyan (light blue)
6 Yellow
7 White

```
PRINT BACKGROUND
7
```

CLEAN

Erases the graphics screen without changing the turtle's position.

CLEARSCREEN
CS

command

Erases the graphics screen, and moves the turtle to its original position at the centre of the screen.

DOT[xy]

command

Leaves a dot at the specified position, co-ordinates [x y]. The turtle does not move and no line is drawn.

Logo will return a message if you ask it to draw a dot outside the limits of the screen.

```
DOT[12 12]
```

FENCE

command

Limits the turtle's movements to the screen boundaries. After using FENCE Logo will not allow you to move the turtle beyond the limits of the screen, neither will it allow you to FENCE a turtle which is already off the screen.

See WINDOW, WRAP

```
FENCE BK 1000

Turtle out of bounds
```

FORWARD n
FD n

command

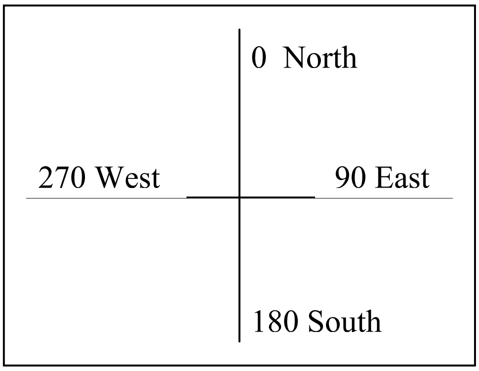
The turtle moves forward n steps without changing its direction. (The turtle will move backwards if n has a minus value.)

```
FD 20
```

HEADING

operation

Outputs a number between 0 and 359, showing the direction in which the turtle is facing. Logo follows the compass system where 0 is north (top of the screen), east 90, south 180 and west 270. When you start Logo, or after you type CS, the turtle's heading is 0.



```
CS
LEFT 1
PR HEADING
359
```

HIDETURTLE
HT

command

Makes the turtle disappear, though it will still draw; a turtle that is hidden will draw faster than one that is visible.

HOME

command

Moves the turtle to the centre of the screen to its origin [0 0]. If the turtle's pen is down, it draws a line from the current position to the origin. The turtle's heading will always become 0.

```
SETPOS [50 1003
HOME
```

LEFT n
LT n

command

The turtle pivots n degrees to the left without changing its position.

```
LT 90
```

PENCOLOUR
PC

operation

Returns a number specifying present pen colour. When you start your Logo, the pencolour is 0.

```
PRINT PC
```

PENDOWN
PD

command

Lowers the turtle's pen so that a line is drawn when it moves.

See also PENUP.

PENERASE
PE

command

The turtle erases any previously drawn lines it passes over.

PENDOWN, PENUP or PENREVERSE cancel the effect of PENERASE.

```
FD 25
PENERASE
BK 50
```

PENREVERSE
px

command

Puts the reversing pen down: when the turtle moves, it draws where there aren't lines and erases where there are.

PENDOWN, PENUP or PENERASE cancel the reversing pen.

```
FD 25
PENREVERSE
BK 50
```

PENUP
pu

command

Lifts the turtle's pen so that no line is drawn when it moves.

```
PENUP FD 50
```

POSITION
pos

operation

Returns the turtle's position as screen co-ordinates [xy]. See DOT.

```
RIGHT 90 FORWARD 5-0
PRINT POSITION
0 50
```

RIGHT n
RT n

command

The turtle pivots n degrees to the right without changing its position.

```
RT 90
```

SCRUNCH

operation

Returns the aspect ratio, [x y], the ratio of the size of a vertical turtle step to the size of a horizontal one. See SETSCRUNCH.

SETBG n **command**

Sets the background colour to the colour n: see BACKGROUND for the table of values for n.

```
SETBG 2
```

SETBORDER
SETBR **command**

Sets the border colour to the colour n: see BACKGROUND for the table of values for n.

```
SETBR 2
```

SETHEADING n
SETH n **command**

Sets the heading of the turtle (turns it) so that it is facing in the direction indicated by the number of degrees n. When the turtle is in its original position (facing the top of the screen) its heading is 0.

SETPC n **command**

Sets the turtle's pen colour to the colour n: see BACKGROUND for the table of values for n. Remember that in the SPECTRUM the INK colour (pencolour) is handled at a character area level and not at a pixel level. A line drawn in a new colour will therefore change the colour of all the lines in the character areas through which it passes.

```
SETPC 6
```

SETPOS [x y] **command**

Given a list of two numbers (x and y coordinates), the turtle moves to that position. If the pen is down, the turtle leaves a trace.

```
FD 50
RT 90
FD 50
SETPOS [0 0]
```

SETSCRUNCH [X Y] **command**

Sets the aspect ratio to [X Y].

If your squares and circles look more like rectangles and ellipses, this command will change the scales on which your images are drawn.

The 'normal' Logo screen is [100 100]. SETSCRUNCH [100 50] will halve the height of a drawing without affecting its width. SETSCRUNCH [50 50] will halve the width as well.

```
TO SQUASH
SETSCRUNCH SE 100 :Y
MAKE "Y:Y -10
REPEAT 20 [FD 30 RT 18]
END
```

```
MAKE "Y 100
REPEAT 10 [SQUASH3 SETSCRUNCH
[100 100]
```

SETX n

command

Moves the turtle to the point n of the x-coordinate while keeping the same y-coordinate. If the pen is down it will draw a horizontal line.

SETY n

command

Moves the turtle to the point n of the y-coordinate while keeping the same x-coordinate. If the pen is down it will leave a vertical line.

SHOWNP

operation

Outputs TRUE if the turtle is in SHOWTURTLE mode, FALSE if it is not.

SHOWTURTLE
ST

command

Makes the turtle visible; see HIDETURTLE.

TOWARDS [x y]

operation

Returns the heading which would be necessary for the turtle to have, if it is to face towards the position [x y]. Note that the turtle is unaffected by using TOWARDS.

```
CS
DOT [10 10]
PR TOWARDS [10 10]
45
```

WINDOW

command

Enables the turtle to move outside the screen area. The screen is like a window viewing only a small portion in the centre of the entire field. When the turtle is in WINDOW mode, it will continue to obey instructions even if it cannot be seen.

The turtle may move up to 32767 steps in any direction from the centre.

See FENCE, WRAP.

```
CS
WINDOW
RT 45
FD 500
PR POS
```

WRAP

command

Makes the turtle's field wrap around the edges of the screen. When the turtle crosses a screen boundary, it immediately reappears on the opposite side.

See FENCE, WINDOW.

```
CS
WRAP
RT 45
FD 500
PR POS
```

XCOR

operation

Returns the x-coordinate of the current position of the turtle.

YCOR

operation

Returns the y-coordinate of the current position of the turtle.

Chapter 3
Words and lists

INTRODUCTION

Words and lists are objects in Logo. In this chapter we will look at the primitives which are useful for manipulating words and lists. A word is a series of characters.

Here are some samples of Logo words:

```
Hello
X
XYZ
XYZ.12.3
MICKEYMOUSE
MICKEY.MOUSE
Good-bye!
```

Each character in a word is called an element. MICKEYMOUSE has 11 elements and MICKEY.MOUSE has 12. The limits of a word are marked by spaces, or by one of the following signs:
before a word

: or" immediately preceding a word (ie, no space between : or" and the word)

after a word

[] () <> + - * /

"followed by a space is known as an empty word.

To treat any of these characters as a normal character, put a / (backslash) (SYS D) before it. This tells Logo to interpret the characters that follows literally as a character, rather than keeping some special meaning it might have. For instance, suppose you wanted to use 3[A]B as a single word. You must type 3 \ [A \ ] B in order to avoid Logo's usual interpretion of the brackets as the envelope around the list. Alternatively, you can give Logo the instruction shown over page.

```
PR "TOPSY \ TURVEY
TOPSY TURVEY
```

(a single word containing a space)

A *list* is composed of words, other lists or both, written within square brackets.

Here are some examples of Logo lists:

```
[HELLO, AGAIN!]
[X2 + Y2 = 223
[MICKEYMOUSE]
[THIS IS A LIST [CONTAINING A
  LIST [[      ]]]
[]
[THIS IS A LIST [CONTAINING
  A LIST][      ]]
```

This contains six elements:

1 : THIS
2 : IS
3 : A
4 : LIST
5 : [CONTAINING A LIST]
6 : []

Element 6 is an empty list.

The primitives that refer to words and lists will now be detailed:

ASCII character **operation**

Returns the ASCII code for the given character. See CHAR.

A list of codes is given in Appendix 2.

Try:

```
PR ASCII "A
65

PR ASCII "B
66
```

```
TO SECRETCODE :WD
IF EMPTYP :WD [OUTPUT CHAR 32]
OUTPUT WORD CODE FIRST :WD SE-
CRETCODE BF : WD
END

TO CODE :LETTER
MAKE "NUM (ASCII :LETTER) + 3
IF :NUM > ASCII "Z [MAKE "NUM
:NUM - 26]
OUTPUT CHAR :NUM
END

PRINT SECRETCODE "CAT
FDW
```

BUTFIRST object
BF object

operation

Returns everything EXCEPT the first element of the specified object (word or list). BUTFIRST of the empty word or list returns an error.

```
PR BUTFIRST [QUEEN ELIZABETH]
ELIZABETH

PR BF "FUNNY
UNNY

PR BF [FUNNY]
```

Note that there is no output

```
PR BF [3
BF doesn't like [] as input

TO TRIANGLE :OBJECT
IF EMPTYP :OBJECT [STOP3
PRINT :OBJECT
TRIANGLE BF :OBJECT
END
TRIANGLE "TRIANGLE
TRIANGLE
RIANGLE
IANGLE
ANGLE
NGLE
GLE
LE
E
```

```
TRIANGLE [MANY GOOD PEOPLE
 LAUGH]
MANY GOOD PEOPLE LAUGH
GOOD PEOPLE LAUGH
PEOPLE LAUGH
LAUGH
```

BUTLAST object operation
BL object

Returns everything EXCEPT the last element of the specified object (word or list).

```
PR BUTLAST [QUEEN ELIZABETH3
QUEEN
```

```
PR BL "FUNNY
FUNN
```

```
PR BL [FUNNY]
```

Note that there is no output

```
PR BL []
```

BL doesn't Like [ ] as input

```
TO BOAST :WD
PR SENTENCE [YOU ARE3 :WD
PR SENTENCE [I AM3 WORD
BUTLAST :WD "IER
END
```

```
BOAST "PRETTY
YOU ARE PRETTY
I AM PRETTIER
```

CHAR n operation

Returns the character whose ASCII code is n, an integer between 32 and 165. You'll receive an Logo message if n is not a valid ASCII code. A list of codes is given in Appendix 2.

COUNT object operation

COUNT returns the number of elements in the specified object (word or list).

See ITEM.

```
PR COUNT [1 2 3 4 5 6 7]
7

PR COUNT [HOW MANY ROADS MUST
A MAN WALK DOWN?]
8

PR COUNT "PEACOCK
7

MAKE "PERSON [HEAD ARMS
     LEGS BODY]
PR COUNT :PERSON
4

TO PICK:INFO
PRITEM (1 +RANDOM COUNT :INFO)
:INFO
END

PICK:PERSON
ARMS
```

EMPTYP object operation

Returns TRUE if the Logo object is empty; otherwise FALSE.

```
MAKE "A []
PR EMPTYP :A
TRUE
MAKE "A 1
PR EMPTYP :A
FALSE

TO LIFE :PERSON :ACTION
IF EMPTYP :PERSON [STOP]
PR SENTENCE FIRST :PERSON
 FIRST :ACTION
LIFE BF :PERSON BF :ACTION
END
LIFE [ALAN LIZ FIONA TIM]
[SINGS LAUGHS WHISTLES SHOUTS]
ALAN SINGS
LIZ LAUGHS
FIONA WHISTLES
TIM SHOUTS
```

```
TO REVPRINT :THING
IF EMPTYP :THING [PR [] STOP]
TYPE LAST :THING
IF LISTP :THING [TYPE CHAR 32]
REVPRINT BL :THING
END

REVPRINT "ELEPHANT
TNAHPELE

REVPRINT "OTTO
OTTO
```

EQUALP objecti object2 operation

Returns TRUE if object and object2 are equal numbers, the same word or identical lists.

```
 PR EQUALP "MARY FIRST [MARY
  JANE]
 TRUE

 PR EQUALP 10 21/3
 FALSE

 TO PRESENCE :OBJECT1 :OBJECT2
 IF EMPTYP :OBJECT2 [OUTPUT
  "FALSE]
IF EQUALP :OBJECT1 FIRST
 :OBJECT [OUTPUT "YES]
OUTPUT PRESENCE :OBJECT1 BF
 :OBJECT2
 END

 PRINT PRESENCE "E "HELEN
 YES

 PR PRESENCE 3 3
 YES

 PR PRESENCE "HELLO "GREETINGS
 FALSE
```

FIRST object operation

Returns the first element of the word or list. FIRST of a word is a character, FIRST of a list may be a word or a list.

```
PR FIRST "HAPPY.NEW.YEAR
H
PR FIRST [HAPPY NEW YEAR3
HAPPY

TO SPELL :WD
IF EMPTYP :WD [STOP]
PR FIRST :WD
SPELL BF :WD
END

SPELL "MOUSE
M
O
U
S
E
```

FPUT object list operation

Returns a new list which is formed by putting the object at the beginning of the list (First PUT).

```
PR FPUT "EENY [MEENY MINEY M0]
EENY MEENY MINEY MO
```

ITEM n operation

Outputs the nth ITEM of a list when given the list and an input number n, provided that the list contains n or more items.

```
PR ITEM 4 [IS THE SKY BLUE?]
BLUE?

PR ITEM 6 [EENY MEENY MINEY MO]
Not enough items in [EENY MEE-
NY MINEY MO]
```

LAST object operation

Returns the last element of a list or the last character of a word.

```
PR LAST [APPLES PEACHES PEARS]
PEARS
```

```
TO REVERSE.WORD :WD
IF EMPTYP :WD [STOP]
PRINT LAST :WD
REVERSE.WORD BUTLAST :WD
END

REVERSE.WORD "CHOCOLATE
E
T
A
L
0
C
0
H
```

LIST objecti1 object2
(LIST object1 object2
 ... objectn)

operation

Returns a list where the elements are object1, object2 etc.

```
MAKE "LINE LIST [ONE] [TW0]
PR :LINE
[ONE3 [TW0]

MAKE "LINE (LIST [ONE] [TW0]
 [THREE])

PR :LINE
ONE3[TW03[THREE]
```

LISTP object

operation

Returns TRUE if object is a list, otherwise FALSE. Note: an empty list is no longer a list; it is taken to bean empty word.

```
PR LISTP [6]
TRUE

PR LISTP 6
FALSE

PR LISTP [CATS AND DOGS]
TRUE
```

```
PR LISTP BF [CATS]
FALSE
```

LPUT object list

operation

Returns a new list which places the object at the End of the list (LastPUT).

```
PRINT LPUT "GERBIL [HAMSTER
 PIG]
HAMSTER PIG GERBIL
PRINT LPUT [CAT MOUSE] [FAT
 HOUSE]
FAT HOUSE [CAT MOUSE]
```

MEMBERP object list

operation

Returns TRUE if the object is an element of the list; otherwise FALSE.

```
PR MEMBERP "L [AB L Y Z]
TRUE
```

As in this case [L] is itself a list, although its sole element is an L.

```
PR MEMBERP "L [AB [L3 Y Z]
FALSE

PR MEMBERP "PIN [S PIN DLE]
TRUE

PR MEMBERP "PIN [SPINDLE]
FALSE

TO VOWEL :LETTER
OUTPUT MEMBERP :LETTER [A E I
 0 U]
END

PR VOWEL "I
TRUE
```

```
PR VOWEL "P
FALSE
```

NUMBERP object operation

Returns TRUE if the object is a number; otherwise FALSE.

```
PR NUMBERP 3
TRUE
```

```
PR NUMBERP [33
FALSE
```

The object is the list [3], hence it is not a number in Logo terms.

```
PR NUMBERP "12:00
FALSE
```

Here, the object is a word.

SENTENCE object1 object2 operation
(SE object 1
object2 ... objectn)

Returns a list composed of the objects in the input.

```
PR SENTENCE "GREEN "APPLES
GREEN APPLES
```

```
PR SE [GREEN] [APPLES]
GREEN APPLES
```

```
TO SCHOOL.LESSON :NAME
PR SE :NAME [PROMISES3
PR SE :NAME [WILL NOT INTER
 RUPT THE TEACHER]
END
```

```
SCHOOL.LESSON "CLAUDINE
CLAUDINE PROMISES
CLAUDINE WILL NOT INTERRUPT
THE TEACHER
```

Remember that same primitives require a list, such as SETPOS [x y], as their input. It is illegal in Logo to enter:

```
SETPOS [:A :B]
```

but you use instead:

```
SETPOS SE :A :B
```

WORD word1 word2
(WORD word1 word2... wordn)

operation

Returns a word consisting of the inputs. WORD does not take a list as an input.

```
PR WORD "FRI "DAY
FRIDAY

PR (WORD "ASTON "ISH "ING)
ASTONISHING

PR WORD "ASTON [ISH]

WORD doesn't like [ISH] as
Input

TO TRIPLE :X
OUTPUT WORD :X WORD :X :X
END

PR TRIPLE "HA
HAHAHA
```

WORDP object

operation

Returns TRUE if the object is a word; otherwise

```
FALSE.
PR WORDP "123ABC
TRUE

PR WORDP 3
TRUE

PR WORDP [ROCKET]
FALSE

PR WORDP "
TRUE

PR WORDP []
FALSE
```

This chart compares four primitives that combine words and lists:

operation	input 1	input 2	output
FPUT	"COW	"HORSE	Logo message
LIST	"COW	"HORSE	[COW HORSE]
LPUT	"COW	"HORSE	Logo message
SENTENCE	"COW	"HORSE	[COW HORSE]
FPUT	"LOGO	[IS WONDERFUL]	[LOGO IS WONDERFUL]
LIST	"Logo	[IS WONDERFUL]	[LOGO [IS WONDERFUL]]
LPUT	"LOGO	[IS WONDERFUL]	[IS WONDERFUL LOGO]
SENTENCE	"LOGO	[IS WONDERFUL]	[LOGO IS WONDERFUL]
FPUT	[THE FOX]	[LOOKS AT FIDO]	[[THE FOX] LOOKS AT FIDO]
LIST	[THE FOX]	[LOOKS AT FIDO]	[[THE FOX][LOOKSATFIDO]]
LPUT	[THE FOX]	[LOOKS AT FIDO]	[LOOKS AT FIDO [THE FOX]]
SENTENCE	[THE FOX]	[LOOKS AT FIDO]	[THE FOX LOOKS AT FIDO]
FPUT	"COMPUTERS	[]	[COMPUTERS]
LIST	"COMPUTERS	[]	[COMPUTERS[]]
LPUT	"COMPUTERS	[]	[COMPUTERS]
SENTENCE	"COMPUTERS	[]	[COMPUTERS]

Chapter 4

Variables

INTRODUCTION

In Logo, variables are created either by using the primitive MAKE or by assigning undefined inputs to a procedure name. A variable can be thought of as a container for a Logo object. This object is known as the variable's *value*.

MAKE name object

Assigns the value 'object' to the variable 'name'. You can also consider a variable as a symbol (or pointer) referring to the object.

Assigning a single number:

```
MAKE "AGE 20
PR :AGE
20
```

Assigning a list of numbers:

```
MAKE "PRICES [10 20 30]
PR :PRICES
10 20 30
```

Assigning a word:

```
MAKE "ANIMAL "FROG
PR :ANIMAL
FROG
```

Assigning a list:

```
MAKE "COLOURS [RED WHITE BLUE]
PR :COLOURS
RED WHITE BLUE
```

Assigning an input list using RL:

```
TO WEATHER
PR [HOW IS THE WEATHER TODAY?]
MAKE "RESPONSE RL
```

```
IF :RESPONSE = CRAINY] [PR [I HOPE
IT STOPS SOON]
IF :RESPONSE = [SUNNY] CPR [GREAT!
I HOPE IT CONTINUES]
END
```

```
WEATHER
HOW IS THE WEATHER TODAY?
SUNNY
GREAT! I HOPE IT CONTINUES
```

```
WEATHER
HOW IS THE WEATHER TODAY?
RAINY
I HOPE IT STOPS SOON
```

It is also possible to assign a value to a variable that is itself a value of another variable thereby building a 'tree'.

```
MAKE "ANIMAL "CAT
PR :ANIMAL
CAT
```

```
MAKE :ANIMAL "KITTEN
PR :CAT
KITTEN
```

(See Chapter 1 for a discussion on global and local variables.)

NAMEP object operation

Returns TRUE if the object has a value; otherwise FALSE.

```
PR NAMEP "FRUIT
FALSE
```

```
MAKE "FRUIT "APPLE
```

```
PR :FRUIT
APPLE
```

```
PR NAMEP "FRUIT
TRUE
```

THING name operation

Returns the contents of name. THING "X is the same as :X; but whereas THING :X is legal, ::X is not!

```
MAKE "MARY "HAPPY
MAKE "HAPPY [A BIRTHDAY PARTY]

PRINT THING "MARY
HAPPY

PRINT :MARY
HAPPY

PRINT THING :MARY
A BIRTHDAY PARTY

TO INC :X
MAKE :X 1 + THING :X
END

MAKE "TOTAL 7
PRINT :TOTAL
7

INC "TOTAL
PR :TOTAL
8

INC "TOTAL
PR :TOTAL
9
```

Chapter 5
Arithmetic operations

INTRODUCTION

Logo understands both integers and decimal fractions.
6 is an integer
.43 is a decimal fraction.

Logo permits you to carry out many arithmetic operations. You may add, subtract, multiply or divide; you may also find arctangents, sines, cosines, tangents and square roots, and test whether a number is greater than, less than or equal to another number.

Certain arithmetic operations always return integers: INT, RANDOM, ROUND.

```
PR DIV 7 2
3.5

PR 7/2
3.5

PR 4.5 + 5.5
10

PR 6 + 4
10
```

Addition, subtraction, multiplication, and division can be used *infix* form. This means that the operator (+ - * /) goes between the inputs. Addition, division and multiplication can also be used in prefix form in which case SUM, DIV or PRODUCT are followed by the two inputs:

```
PR PRODUCT 4 4
16
```

The primitive EQUALP, described in Chapter 2, is often used with arithmetic operations. The INFIX operation = is equivalent to EQUALP.

We will now list the primitives concerned with arithmetic.

ARCCOS n

operation

Returns the value, in degrees, of the arccosine of n.

```
PR ARCCOS 0.45
63.256316
```

ARCCOT n

operation

Returns the value, in degrees, of the arc cotangent of n.

```
PR ARCCOT 1
45
```

ARCSIN n

operation

Returns the value, in degrees, of the arcsine of n.

```
PR ARCSIN 0.45
26.743684
```

ARCTAN n

operation

Returns the value, in degrees, of the arctangent of n.

```
PR ARCTAN 1
45
```

Arcsines and arccosines may be found as follows:

```
TO ARCSINE :X
OUTPUT ARCTAN :X /
(SORT 1- :X * :X)
END
```

```
TO ARCCOSINE :X
OUTPUT ARCTAN
SQRT 1 -
:X * :X)/:X
```

COSINE n
COS n

operation

Returns the value, in degrees, of the cosine of n

```
PR COS 60
0.5
PR COS 30
.8660254
```

COTANGENT n

COT n

operation

Returns the value, in degrees, of the cotangent of n.

```
PR COT 45
```

DIV a b

operation

Returns the quotient obtained by dividing a by b.

```
PR DIV 24 2
12

PR DIV -24.24 2
-12.12

PR DIV -25 0
Can't divide by zero
```

INT n

operation

Returns the INTeger portion of n by removing any decimal fractions; see ROUND.

```
PR INT 5.2129
5

PR INT 5.5
5

PR INT -5.5
-5
```

Numbers may be tested to see if they are integers:

```
TO INTP :N
IF NOT NUMBERP :N [OP[NOT A N
UMBER]]
OP  :N  =  INT  :N
END
```

```
PR INTP 17
TRUE

PR INTP 100/8
FALSE

PR INTP "ONE
NOT A NUMBER
```

PRODUCT a b
(PRODUCT a b... n)

operation

Returns the product of the inputs. It is equivalent to the INFIX operation *. If PRODUCT has more than two inputs, parentheses must appear around PRODUCT and its inputs.

```
PR PRODUCT 5 5
25

PR (PRODUCT 5 5 2)
50

TO SQUARE :X
PR PRODUCT :X :X
END

SQUARE 2
4

TO CUBE :X
PR (PRODUCT :X :X :X)
END

CUBE 2
8
```

RANDOM n

operation

If n is a positive integer, returns a random number between 0 and (n-1).

Random 6 could output 0,1,2, 3,4 or 5

```
TO DICE
OUTPUT 1 + RANDOM 6
END
```

```
PR DICE
3
```

Note carefully that

```
1 + RANDOM 6
```

is used in this example, because

```
RANDOM 6+1
```

does not give the correct answer. Alternatively brackets can be used to 'collect' true correct terms together:

```
(RANDOM 6) + 1
```

REMAINDER a b operation

Returns the remainder left when a is divided by b.

```
PR REMAINDER 16 4
0

PR REMAINDER 16 5
1

TO EVENP :NUM
OUTPUT 0 = REMAINDER :NUM/2
END

PRINT EVENP 5
FALSE

PR EVENP 2
TRUE
```

The following procedure tells whether the first input is a divisor of the second:

```
TO DIVISORP :A :B
OP 0 = REMAINDER :B :A
END

PR DIVISORP 3 5
FALSE
```

ROUND n

operation

Returns n rounded to the nearest integer. Compare these examples with INT.

```
PR ROUND 5.219
5

PR ROUND 5.5
6

PR ROUND -5.5
-6
```

SINE n
SIN n

operation

Returns the value, in degrees, of the sine of n.

```
PR SIN 30
0.5
```

SQRT n

operation

Returns the square root of n; n must be positive.

```
PR SORT 49
7

PR SQRT 4567
67.5796
```

The procedure DISTANCE takes any two positions as inputs, and outputs the distance between them:

```
TO DISTANCE :POS1 :POS2
OP SQRT SUM SQ (FIRST
:POS1) -
(FIRST :POS2) SQ (LAST
:POS1) -
(LAST :POS2)
END

TO SQ :N
OP :N * :N
END

PR DISTANCE [-70 103 [50 60]
130
```

SUM a b **operation**

(SUM a b...n)

Returns the sum of the inputs a b. It gives the same result as IN FIX operation +. If SUM has more than two inputs, parentheses must appear around SUM and its inputs

.

```
PR SUM 5 21
7
```

```
PR (SUM 523)
10
```

TANGENT n **operation**

TAN n

Returns the value, in degrees, of the tangent of n.

```
PR TAN 50
1.1917536
```

INFIX OPERATIONS

Avoid confusion between a negative number and the INFIX operation - (subtraction). It is good practice to put a space both before and after the sign unless you are giving a negative number as input. Examine the examples carefully!

A word is usually separated from the element which comes before, and that which comes after, by spaces.

There are certain other delimiters:
[ ]( ) = < > + - * /

a + b **INFIX operation**

plus

Returns the sum of the inputs a and b.

```
PR 5 + 2
7
```

```
PR-5+2
-3
```

a-b **INFIX operation**

minus

Returns the difference between the inputs.

```
PR 7 - 1
6
```

```
PR-7-1
-8

PR -7 - -2
-5
```

a * b

INFIX operation

Returns the product of a and b (a * b).

```
PR 6 * 2
12

PR 6 * -2
-12

PR 2 + 3 * 4
14

PR (2 + 3) * 4
20
```

a / b

INFIX operation

Returns the dividend of a and b (a divided by b).

```
PR 6 / 6
1

PR-6/6
-1

PR 6 / 0
Can't divide by zero
```

a < b

NFIX operation

Returns TRUE if a is less than b; otherwise FALSE.

```
PR 2 < 3
TRUE

PR 3 < 3
FALSE

PR 3 < "TOTAL
< doesn't Like TOTAL as input
```

a > b

NFIX operation

Returns TRUE if a is greater than b; otherwise FALSE.

```
PR 4 > 3
TRUE
```

a = b

IN FIX operation

Returns TRUE if a is equal to b, whether the inputs are words, lists or numbers; otherwise FALSE. Equivalent to the PREFIX operation EQUALP.

```
PR 80 = 100 - 20
TRUE

PR 80 = 100 -20
FALSE
You don't say what to do
with –20

PR 80 = (100 - 20)
TRUE
```

Chapter 6
Defining and editing

INTRODUCTION

There are two ways of defining procedures. The usual method is to use the edit mode, ie, the Logo Editor, although the use of the TO mode is perfectly acceptable when defining simple procedures. The advantage of using the Editor is of course that you can edit any mistakes you make immediately.

EDIT MODE
EDIT procedures
ED procedures

To enter edit mode, type EDIT or ED followed by the name, or list of the procedures to be edited.

```
?ED "SPI
```

If you have not defined the procedure, your screen will look like this:

```
TO SPI

LOGO EDITOR © SOLI / LCSI C
```

If you have already defined the procedure SPI, your screen might read:

```
TO SPI :SIDE :ANGLE
FD :SIDE
RT :ANGLE
SPI :SIDE + 5 :ANGLE
END

LOGO EDITOR © SOLI / LCSI C
```

If you type ED or EDIT, and do not follow it with anything, Logo will bring you whatever was last in the Editor. ED [ ] or EDIT [ ] will always bring you an empty Editor.

When in the edit mode the prompt character ? does not appear. Instead, the cursor shows where you are typing; it can be moved anywhere in the text by

using the special keys listed below.
Characters can be inserted or deleted.

When you press the ENTER key, the cursor moves to the next line and waits for you to type. A new line is inserted if a line is already there.

A space or a letter is inserted wherever you type one or the other.

Logo does not execute instructions when in edit mode.

While in edit mode, you may define one, or more, procedures. Each procedure must have its own title line - TO, procedure name, inputs - and its own END line.

EDITING KEYS

Note: CAPS refers to the CAPS SHIFT key.
SYS refers to the SYMBOL SHIFT key.
Keys must be pressed simultaneously.

Moving the cursor	
CAPS 5	Moves cursor one character to left.
CAPS 6	Moves cursor one line down.
CAPS 7	Moves cursor one line up.
CAPS 8	Moves cursor one character to right.
CAPS 0	Deletes one character to left.
E MODE CAPS 5	Moves cursor to beginning of line.
E MODE CAPS 6	Moves cursor to end of screen.
E MODE CAPS 7	Moves cursor to beginning of screen.
E MODE CAPS 8	Moves cursor to end of line.
E MODE B	Moves cursor to beginning of text.
E MODE E	Moves cursor to end of text.
Deleting and inserting	
E MODE Y	Deletes the line from the position of the cursor onwards and saves it.
E MODE R	Enters the 'saved line' at the position of the cursor.
Scrolling	If your text is longer than one screen page, scrolling allows you to move from one page to the next or to a previous one.

SYS S

Tells Logo to stop scrolling; press any key to make it start again.

E MODE P

Moves cursor to previous page.

E MODE N

Moves cursor to next page.

EXITING FROM EDIT MODE

You have two options when leaving edit mode. If you wish Logo to incorporate the modifications you have just made, use EMODEC. If you don't wish to modify, use CAPS BREAK/SPACE.

E MODE C

Press CAPS SHIFT and SYS simultaneously until the E appears in the lower right corner of the screen. Let go of both keys and press C. Logo will incorporate all the modifications you have made and tell you which procedures have been defined during that editing session.

CAPS BREAK/SPACE

Press CAPS and BREAK/SPACE simultaneously. Logo will leave the Editor and ignore any changes you have made.

When outside edit mode, you may use any of the editing keys as long as you remain within one Logo line.

EDNS name **command**
EDNS (name list]

Allows you to edit names and their values. With no inputs, Logo will list all variables and their values. With an input, Logo will list all the variables named. In leaving the Editor, Logo will interpret all the MAKE commands you have just typed so that they contain their new values.

EDNS

Your screen will show

```
MAKE "COLOURS [RED WHITE AND
 BLUE]
MAKE "NAMES CTOM DICK AND
 HARRY3
MAKE "NUMBER 55
```

Edit the values

```
MAKE "COLOURS [ORANGE AND PINK]
MAKE "NAMES "CLAUDINE
MAKE "NUMBER 16
```

Enter E mode and press C to leave the Logo Editor. Now type

```
PONS
MAKE "COLOURS [ORANGE AND PINK]
MAKE "NAMES "CLAUDINE
MAKE "NUMBER 16
```

TO MODE command
TO name input1 ... inputn

TO mode enables you to define a procedure of your own. To enter TO mode, type TO followed by the (unique) name of your procedure.

```
?TO SQUARE :SIDE
>REPEAT 4 CFD :SIDE RT 90]
>END
SQUARE Defined
?
```

TO tells Logo to enter the TO mode. Logo does not carry out instructions when in TO mode; it remembers them. It indicates you are defining a procedure, with a given name and specified inputs.

The prompt changes from ? to >.
END must be the only word on the last line.

If you decide to abandon a procedure you have started, press CAPS BREAK/SPACE. To change the definition of a procedure you may either:
  use the ERASE procedure and redefine the procedure,
or:
enter Edit mode and use the Logo Editor.

END

END has a special status. END is required when defining a procedure. It tells Logo that you have finished defining the procedure, and it must be by itself on the last line.

You must use END to separate procedures when you define more than one in the Editor, but it is not necessary to define the END of the last procedure.

Chapter 7
Conditional expressions and flow of control

INTRODUCTION

Within a procedure, Logo reads and carries out your instructions line by line. If one of your instructions is a *subprocedure*, Logo will execute all the commands in that subprocedure *before* going on to the next command of the *superprocedure*. The order in which Logo reads and follows your instructions can be modified; let's see how.

Conditional instructions tell Logo to execute a particular action if a particular condition is true.

Repeat instructions tell Logo to execute a list one or more times.

The STOP instruction tells Logo to STOP the current procedure without continuing to the END but to continue with the next part of the superprocedure.

BYE **command**

Exits from Logo and returns control to BASIC. Note: Logo can be restarted by typing RUN.

IF pred instructionlist1 instructionlist2 **command or operation**

The first input to IF is a *predicate*, or logical operation, that IF tests. Predicates are operations that return TRUE or FALSE. If the predicate is TRUE, instructionlisti is executed. If the predicate is FALSE, instructionlist2 is executed if it is present.

The procedures CHOOSE and CHOOSE1 show the use of IF as a command, firstly with two inputs and secondly with three:

```
TO CHOOSE
IF 0 = RANDOM 3 [OP "YES]
OP "NO
END
```

```
PR CHOOSE
YES

TO CHOOSE1
IF 0 = RANDOM 3 [OP "YES][OP "No]
END

PR CHOOSE1
NO
```

CHOOSE2 shows the use of IF as an operation:

IF as an OPERATION

```
TO CHOOSE2

OP IF 0 = RANDOM 3 ["YES3] ["N0]
END

PR CHOOSE2
YES
```

OUTPUT object
OP object

command

This primitive, like STOP, tells Logo to STOP the current procedure without continuing to the END. But, unlike STOP, the OUTPUT primitive passes its object back to the calling procedure for possible use.

A procedure that terminates during execution with END acts as a command procedure, whereas a procedure terminating with OUTPUT acts as an operation.

```
TO BIRTHDAY.SONG
OP [HAPPY BIRTHDAY]
END

PR SE BIRTHDAY.SONG [TO YOU]
HAPPY BIRTHDAY TO YOU

TO MEAN :X :Y
OUTPUT (:X + :Y)/2
END
PR MEAN 6 3
4.5
```

```
TO DECIDE :LETTER :WD
IF EMPTYP :WD [OUTPUT "FALSE]
IF EQUAL- :LETTER FIRST :WD
[OP "TRUE] [OP DECIDE :LETTER
 BF :WD]
END

PR DECIDE "H "HOUSE
TRUE
PR DECIDE "X "HOUSE
FALSE
```

REPEAT n instructionlist command

Repeats a list of instructions n times; n must be positive; a decimal fraction is truncated to an integer.

```
REPEAT 4 [FD 40 RT 90]
REPEAT 2 [PR [HIP HIP HOORAY!]]
REPEAT 360 [FD 1 RT 1]
```

RUN instructionlist command

With a Logo list as input, RUN executes the list as a Logo line.

```
RUN [PRINT [GOOD MORNING]]
GOOD MORNING

RUN LIST "PRINT [GOOD MORNING]
GOOD MORNING

MAKE "ORDER "PRINT
MAKE "INPUT [GOOD MORNING]

RUN LIST :ORDER :INPUT
GOOD MORNING

TO CALCULATE
PR RUN READLIST
END

CALCULATE
3+3
6
```

```
CALCULATE
42 = 7 * 8
FALSE

CALCULATE
REMAINDER 12 5
2

TO MAP :COM :LIS
IF EMPTYP :LIS [STOP]
RUN LIST :COM FIRST :LIS
MAP :COM BF :LIS
END

TO SQUARE :SIDE
REPEAT 4 [FD :SIDE RT 90]
END

MAP "SQUARE [10 20 30 40 50]
```

STOP command

Stops the execution of the procedure currently running and returns control to the procedure which called it.

```
TO REDUCE :OBJ
IF EMPTYP :OBJ [STOP]
PRINT :OBJ
REDUCE BL :OBJ
END

REDUCE "CHOCOLATE
CHOCOLATE
CHOCOLAT
CHOCOLA
CHOCOL
CHOCO
CHOC
CHO
CH
C
```

```
TO INCREASE :OBJ
IF EMPTYP :OBJ [STOP]
INCREASE BL :OBJ
PR :OBJ
END

INCREASE "CHOCOLATE
C
CH
CHO
CHOC
CHOCO
CHOCOL
CHOCOLA
CHOCOLAT
CHOCOLATE

TO ALTERNATE :OBJ
IF EMPTYP :OBJ [STOP]
PR :OBJ
ALTERNATE BF :OBJ
PR :OBJ
END

ALTERNATE "BCD
BCD
CD
D
D
CD
BCD
```

TOPLEVEL command

When Logo executes the command TOPLEVEL, it immediately stops the command it is evaluating, and returns control to TOPLEVEL. Compare with STOP.

```
TO ALTERNATEl :OBJ
IF EMPTYP :OBJ [TOPLEVEL]
PR :OBJ
ALTERNATEl BF :OBJ
PR :OBJ
END
```

```
ALTERNATE "BCD
BCD
CD
D
```

The most useful way to use TOPLEVEL is in error escapes.

Both procedures give the global variables OBJ the sublist L which begins with :X.

```
TO LOOKFOR :X :L
IF EMPTYP :L [STOP]

IF :X = FIRST :L (MAKE "OBJ) :L]
LOOKFOR :X BF :L
END

?LOOKFOR "Z AZBCZXY
?PR :OBJ
ZXY

TO LOOKFOR1 :X :L
IF EMPTYP :L [STOP]
IF :X = FIRST :L [MAKE "OBJ :L
  TOPLEVEL]
LOOKFOR1 :X BF :L
END

?LOOKFOR1 "Z "AZBCZXY
?PR :OBJ
ZBCZXY
```

In the second example, the procedure returns to TOPLEVEL as soon as OBJ finds the value looked for, Z. Compare with the first procedure which continues running every instruction in the procedure whether or not OBJ has found the value it is looking for.

Chapter 8

Logical operations

INTRODUCTION

Sinclair Logo contains the primitives AND, NOT and OR; they allow the user to perform logical operations.

The inputs to these primitives can only be TRUE and FALSE, which in Logo are special words.

In their turn the primitives AND, NOT and OR produce results that are only the words TRUE and FALSE.

The term predicate is used to describe a function that outputs TRUE or FALSE; hence AND, NOT and OR are considered to require predicates as their inputs.

AND pred1 pred2 (AND pred1 pred2 . . . predn)

operation

Returns TRUE if all the inputs are TRUE, otherwise FALSE. If AND has more than two inputs, parentheses must appear around AND and its inputs.

```
PRINT AND TRUE TRUE
TRUE

PRINT AND FALSE FALSE
TRUE

PRINT (AND TRUE TRUE FALSE)
FALSE

PRINT 16=16
TRUE

PRINT 3=3
TRUE

PRINT AND 16=16 3=3
TRUE
```

```
PRINT AND 16 3
3 is not true or false

TO DECIMAL? :OBJ
OP AND NUMBERP :OBJ CHECK :OBJ
END

TO CHECK :OBJ
IF EMPTYP :OBJ [OP "FALSE]
IF EQUALP FIRST :OBJ ". [OP "-
TRUE]
OP CHECK BF :OBJ
END

PRINT DECIMALP 17.0
FALSE

PRINT DECIMALP 17.635
TRUE
```

NOT pred — operation

Returns TRUE if the predicate is FALSE and FALSE if the predicate is TRUE.

```
PRINT NOT EQUALP "A "Z
TRUE

PRINT NOT EQUALP "E "E
FALSE

PRINT NOT "K=FIRST "KERCHIEF
FALSE
```

OR pred1 pred2
(OR pred1 pred2... predn) — operation

Returns TRUE if any of the predicates is TRUE, otherwise FALSE. If OR has more than two inputs, parentheses must appear around IF and its inputs.

```
PRINT OR TRUE TRUE
TRUE

PRINT OR TRUE FALSE
TRUE
```

```
PRINT OR FALSE FALSE
FALSE

PR OR 16=16 3=3
TRUE

PRINT OR 6 3
3 is not true or false

TO MOUNTAINS
CS RT 45
SUBMOUNTAIN
END

TO SUBMOUNTAIN
FD 5 + RANDOM 10
IF OR YCOR > 50 YCOR < 0 [SETH
180 - HEADING]
SUBMOUNTAIN
END
```

Chapter 9
The outside world

INTRODUCTION

This chapter contains primitives which allow you to use your keyboard in special ways to communicate with Logo; see also Chapter 10.

KEYP

operation

Returns TRUE if a valid key, or combination of valid keys, is being pressed; otherwise FALSE.

The following procedure moves the turtle in small steps. When you press P, the turtle turns RT 30; when you press Q, the turtle moves LT 30.

```
TO STEER
FD 2
IF KEYP [TURN RC]
STEER
END

TO TURN :DIR
IF :DIR = "p [RT 30]
IF :DIR = "Q [LT 30]
END
```

Print object
PR object

command

When the PRINT command is given, followed by ENTER, the outermost brackets or the quote marks are not printed. Compare with SHOW and TYPE.

Note: PRINT causes a linefeed to occur after the printing has been performed.

```
PRINT "A
A

PRINT [YES INDEED]
YES INDEED

TO REPRINT :NOTE :NUM
IF:NUM < 1 [STOP]
PR:NOTE
```

```
REPRINT :NOTE :NUM - 1
END
REPRINT [HAPPY BIRTHDAY TO
 YOU] 4
HAPPY BIRTHDAY TO YOU
HAPPY BIRTHDAY TO YOU
HAPPY BIRTHDAY TO YOU
HAPPY BIRTHDAY TO YOU
```

READCHAR
RC

operation

Waits for the user to press a valid key, or valid combination of keys. The operation returns the appropriate character, but does not print it on the screen.

```
TO DRAW
MAKE "Z READCHAR
IF :Z = 5 [LT 90]
IF :Z = 6 [BK 10]
IF :Z = 7 [FD 10]
IF :Z = 8 [RT 90]
DRAW
END
```

READLIST
RL

operation

Returns a list that you give as input. The whole of the line entered before the ENTER key is pressed is taken as a list. Each character is printed on the screen as it is typed.

```
TO FAIRY-TALE
PR [MIRROR, MIRROR ON THE WALL,
WHO IS THE
FAIREST OF THEM ALL?]
IF RL = [HER HIGHNESS, THE QUEEN]
[PR [THE QUEEN
SMILES] STOP] [PR [THE QUEEN
IS ANGRY AND ASKS THE
QUESTION
AGAIN] FAIRY TALE]
END

FAIRY.TALE
MIRROR, MIRROR, ON THE WALL,
WHO IS THE FAIREST OF
```

```
THEM ALL?
SNOW WHITE
THE QUEEN IS ANGRY AND ASKS
THE QUESTION AGAIN
MIRROR, MIRROR ON THE WALL,
WHO IS THE FAIREST OF
THEM ALL?
HER HIGHNESS, THE QUEEN
THE QUEEN SMILES
```

SHOW object command

Prints the word, list or numbers given as input. Lists are printed with brackets around them. As with PRINT, the command SHOW causes a linefeed to occur after the printing. See also PRINT and TYPE.

```
SHOW "HARRY
HARRY

SHOW [FAIRY TALE]
[FAIRY TALE]

SHOW [A B C]
[A B C]
```

SOUND [duration pitch] command

Allows your Spectrum to make sounds. The duration of the sound is given in seconds, and its pitch in semitones above middle C is given by positive integers; below middle C by negative integers. The first input to SOUND must be between 0 and 255. The second input must be between -62 and 75.

```
SOUND [1 0]
```

This procedure will make each key give you a sound:

```
TO SING
SOUND SE 0.5 (ASCII RC) - 65
SING
END
```

```
TO PLAY :LIST
IF EMPTYP :LIST [STOP]
SOUND SE 0.5 ASCII FIRST :LIST
PLAY BF :LIST
END

PLAY [A B C]
```

STARTROBOT command

Causes subsequent turtle commands FD, BK, PU, PD, RT and LT to be mirrored by the mechanical robot or floor turtle which is attached to your Spectrum.

Note: Logo searches for the binary file containing the instructions to drive the robot. If no binary file is found in memory, it will try to load a file with the name ROBOT from your microdrive cartridge, or from cassette if your 'drive' number is set to zero. (See SETDRIVE.)

STOPROBOT command

Reverses the effect of STARTROBOT.

TYPE object
(TYPE object1 object2... objectn) command

Prints its object on the screen. But, unlike PRINT, there is no linefeed after the printing.

Outermost brackets of a list are not parted.

If TYPE has two or more inputs then TYPE and all the inputs must be enclosed in parentheses.

See also PRINT and SHOW.

```
TYPE "A
A
TYPE [A B C]
ABC
TYPE "A TYPE [A B C]
AA B C

TYPE [A B C][D E F][G H I])
A B CD E FG H I
```

WAIT n command

Logo waits n/50ths of a second.

```
TO SLOW.MARCH :DIST
REPEAT :DIST [FD 1 WAIT 1]
END

HT
REPEAT 4 [SLOW.MARCH 80 RT 90]
```

Chapter 10

Screen commands

INTRODUCTION

This chapter describes primitives which allow you to interact with your computer. When you load your Sinclair Logo you are in textscreen mode; 22 lines are available for text.

When you are in graphics mode, 22 lines are available for graphics, with two more lines at the bottom for text. You immediately enter graphics mode when you give Logo a graphics command.

Each line has space for 32 characters. The first column in Logo mode and in TO mode is a prompt. This indicates that Logo is ready for your instructions. The last column is reserved for an exclamation point which indicates an unfinished Logo line longer than 32 characters.

BRIGHT n

command

BRIGHT 1 tells Logo to start printing on 'bright paper'. Logo will stay in this state until the command BRIGHT 0 is executed, the toplevel procedure is completed, or an error occurs.

```
TO DIRECTION
TS
SETCUR [12 3]
BRIGHT 1
PR [DIRECTION]
BRIGHT 0
END
```

CLEARTEXT
CT

command

Clears all text from the screen. When in graphics mode, CT will clear the two lines at the bottom of the screen.

COPYSCREEN

operation

Copies everything presently on your screen on to your ZX Printer (provided it is connected!) This allows you to make a 'hard copy' of your text and graphics procedures.

```
CS REPEAT 8 [REPEAT 6 [FD 10
 RT 45]
RT 45] HT
COPYSCREEN
```

CURSOR

operation

The cursor is the flashing rectangle on your screen which moves as you type. It indicates where the next character you type will appear. The primitive CURSOR returns the column and line numbers of the cursor position as a list.

```
PR CURSOR
```

FLASH

command

Tells Logo to start printing on 'flashing paper'. Logo will stay in this state until the command NORMAL is executed, the toplevel procedure is completed, or an error occurs.

```
TS
SETCUR [12 10]
FLASH
PR [HOW JOLLY]
```

INVERSE

command

Tells logo to start printing with the background and foreground colours inversed. Logo will stay in this state until the command NORMAL is executed, the toplevel procedure is completed, or an error occurs.

```
TS
SETCUR [12 10]
INVERSE
PR [IS THIS JOLLY TOO?]
```

NORMAL

command

Tells Logo to resume printing without any inversion or flashing. This command has no effect if the printing is already normal.

```
FLASH [IGUANAS ARE CUTE] NORMAL
 [IGUANAS ARE CUTE]
```

OVER n **command**

OVER 1 tells Logo to start printing over any existing lines or text. (The over-printing is made on an exclusive-OR principle: set pixels are reset, and reset pixels are set.)

Logo will stay in this state until the command OVER 0 is executed, the toplevel procedure is completed, or an error occurs.

```
TO OVERWRITE
TS
SETCUR [12 3]
PR [WRITING]
OVER 1
WAIT 15
SETCUR [12 3]
PR [WRITING]
OVER.O
END
```

SETCURSOR [a b]
SETCUR [a b] **command**

Moves the cursor to a position indicated by the two inputs a and b. The first element is the column number; the second, the line number. Columns are numbered from 0 to 30 and lines from 0 to 21.

```
TO MOVECURSOR :X :Y
SETCURSOR LIST (:X + FIRST CURSOR)
(:Y + LAST CURSOR)
END
```

```
SETCURSOR [25 12] TYPE "A MOVE
CURSOR 2 5 PRINT "B
```

SETTC [n n] **command**

Allows the user to specify the background colour and the foreground colour when printing text.

```
SETTC [2 4] PR [GREEN ON RED]
```

See BACKGROUND (Chapter 2) for the table of values for n.

TEXTSCREEN	**command**
TS	With TS your entire Logo screen is available for texts. You cannot see the turtle while you are in text mode.

Chapter 11
Workspace

INTRODUCTION

When you load Logo into your Spectrum, it occupies only part of the memory. The rest of the memory is available to you as your *workspace*.

The primitives presented in this chapter allow you to print out the procedures and variables from the workspace, and to erase them.

Be careful when using the 'erasing' primitives as their effects are permanent!

ERALL

command

ERases ALL that you have created in the workspace. It is as if you turned off and restarted Logo. Be sure that you have saved all the procedures and variables you want to keep before you use this command - see Chapter 12 for details.

Note: The current contents of the editor are not affected by ERALL. To clear the editor as well, use ERALL and EDIT [ ].

ERASE name
ER name

command

Erases the named procedure or procedures from your workspace.

```
ERASE "BOX
ER [BOX]
```

Erases the procedure called BOX

```
ER [TRIANGLE BOX]
```

ERN name

command

Erases the Named variable(s) from your Workspace.

```
ERN "SIDE
```

or

```
ERN [SIDE]
```

Erases the variable SIDE

```
ERN [SIDE ANGLE]
```

Erases the variables SIDE and ANGLE

ERN command

ERases the NameS and values of all variables in your workspace.

ERPS command

ERases all the Procedures from your workspace.

PO name command

Prints Out the definition of the named procedure(s).

```
PO "SQUARE
```

or

```
PO [SQUARE]
```

```
TO SQUARE :SIDE
REPEAT 4 [FD :SIDE RT 90]
END
```

POALL command

Prints Out the titles and definitions of ALL procedures, and the value of every variable, in your workspace.

PONS command

Prints Out the NameS and values of all the variables in your workspace.

```
AND MAKE "SIDE "LENGTH
MAKE "ANGLE 90
MAKE "COLOURS [PINK BLUE]
```

POPS **command**

Prints Out the definitions of all the Procedures currently in the workspace.

POTS **command**

Prints Out the TitleS of all the procedures in the workspace.

```
POTS
TO SQUARE :SIDE
TO GREET :R
```

Note: Use the keys SYS and S to control the scrolling of the screen; if necessary use PRINTON (your print-out primitive) PRINTOFF if you wish your printing to appear on the ZX Printer as well as the screen.

Chapter 12

Saving and retrieving your work

INTRODUCTION

While you are programming in Sinclair Logo, the machine stores all the procedures you have taught it in its workspace. Unfortunately, when you turn the machine off, it forgets everything because the workspace is a part of the computer memory that remembers only while the computer is on.

You may save your work at any time during a Logo session. First, you arrange your work in files; you decide what should go into each file. Then you can save it on a cassette or Microdrive, and retrieve it later when required.

A File can be of three types:
1 Logo procedural file.
A file containing Logo procedures (and variables if created using SAVEALL, see below).

2 Editor file.
The current contents of the Logo Editor can be saved, and retrieved.
3 Screen file.

The current display can be saved, and retrieved. (See Chapter 14 for details of how binary files can be saved and retrieved.)

Note: If at any time when using Logo a BASIC error report appears, as will always occur if, for example, you try to use a Microdrive unit which does not have a cartridge in it, then Logo can be restarted with the BASIC line.

```
RUN
```

When using this `warm` start, the screen display will Be lost but the workspace is returned intact.

Saving your work on cassette

SAVE "filename [names]

You can save your work on cassette files; any cassette tape will do. While not necessary, a tape counter is useful. The cassette recorder needs an input socket for use with a microphone and an output socket for use with earphones. Connect the mic socket on the cassette player to the mic socket on the ZX Spectrum, and if you have the ear socket already connected, pull out the ear jacks.

You may give your file any name you like, precedec by a " (quote mark). Filenames can have up to seven characters. Follow the filename with the name of the procedure to be saved.

```
SAVE "MYFILE "SQUARE1
```

You may save more than one procedure in a file by typing brackets around the procedure names.

```
SAVE "MYFILE CSQUARE1 GREET]
```

Start the tape by simultaneously pressing the PLAY and RECORD keys. Logo will tell you to press any key. While the file is being saved the border flashes Once the flashing has finished you may STOP the tape.

Replace the ear jack.

It's a good idea to keep a written record describing each file. You can save several different files on the cassette. Advance the tape approximately ten counts before saving another file.

SAVEALL "filename — **command**

Logo saves everything currently in the workspace under the given filename, ie, all procedures and variables.

SAVED "filename — **command**

Logo saves everything currently in the Editor under the given filename.

SAVESCR "filename **command**

Logo saves everything currently on the screen under the given filename.

Saving your work on Microdrive

The following primitives apply to the Microdrive.

SETDRIVE 0... 8 **command**

Tells Logo whether you wish to use a cassette player, SETDRIVE (also the default state), or a particular Microdrive, 1-8.

To save your workspace on Microdrive 1 use:

```
SETDRIVE 1
SAVEALL "FILENAME
```

CATALOG **command**

(Microdrive only)

Prints the name of the cartridge, all the filenames (Logo files and others) and the amount of unused space remaining (in K).

`SETDRIVE 1` (for Microdrive 1)
`CATALOG`

ERASEFILE "filename, filetype **command**

(Microdrive only)

Instructs Logo to erase the file named from your Microdrive. If no filetype is entered Logo will assume a filetype of LOG. Other types of file can be
BIN (binary)
SCR (screen)
TXT (editor)

Retrieving your work from cassette

LOAD
"filename, filetype

Position your cassette tape at, or before, the file you want to retrieve. Connect your recorder to the Spectrum in the normal manner for loading, referring to your Introduction to the Sinclair

Start the tape by pressing the PLAY key. When the File is loaded, Logo tells you that the procedures are Defined, and the prompt and cursor reappear on the screen.

```
LOAD "MYFILE
SQUARE1 defined
GREET defined
```

Everything you saved in the file will be loaded back into your workspace.

Note: Several files can be loaded into the workspace, one by one. Be careful, if you use the same procedure name in two or more files then Logo will leave you with the newest procedure.

LOADD "filename **command**

Instructs Logo to load everything that was saved in your SAVED "filename files and make it the current contents of the Editor.

LOADSCR "filename **command**

Instructs Logo to load everything saved in your SAVESCR "filename file and display it.

Retrieving your work from Microdrive

The particular Microdrive that you will use must be the 'current drive'. Use SETDRIVE 1 ... 8 if necessary.

Files may now be retrieved, as from cassette, using:

LOAD "filename
LOADD "filename
LOADSCR "filename

Saving your work on the printer

The following primitives allow a ZX Printer to be used with Sinclair Logo.

PRINTON Tells Logo to print everything that follows. Everything you print on your screen will also be printed on your Printer.

PRINTOFF Tells Logo to stop printing.

COPYSCREEN Tells Logo to copy whatever is on the upper 22 lines of your screen. This primitive works with both the text screen and the graphics screen.

Chapter 13

Definitions and redefinitions of functions

INTRODUCTION

The primitives presented in this chapter enable procedures to be defined and handled from within other procedures.

COPYDEF newname name — command

Copies the definition of an existing procedure name to a new procedure name.

```
COPYDEF "SQ "SQUARE
```

copies the definition of SQUARE to SQ. The existing procedure is not erased.

DEFINE namelist — command

Takes two inputs. The first is the name of a procedure and the second a list. The elements of the list are a list of inputs to the new procedure, and a list for each procedure line. DEFINE allows you to write procedures which define other procedures.

```
DEFINE "TRY [[:X :Y][PRINT
:X][PRINT :Y]
PO "TRY

TO TRY :X :Y
PRINT :X
PRINT :Y
END

DEFINE "GREET [[][PR [GOOD
DAY]]
PO "GREET

TO GREET
PRINT [GOOD DAY]
END
```

Note: If the new procedure is not to have any inputs, then the first item of the list is to be an empty list [ ].
No END is written at the end of the list.

DEFINEDP word

operation

Returns TRUE if a word is the name of a procedure- otherwise FALSE.

PRIMITIVE? word

operation

Returns TRUE if its input is a Logo primitive- otherwise FALSE.

TEXT

operation

When given a procedure name as input, TEXT returns the text of the procedure as a list. The format is described under DEFINE. With TEXT you can write procedures which examine and manipulate other procedures.

```
TO SHAPE :X :Y
FD :X
RT :Y
END

PR TEXT "SHAPE
[:X :Y] [FD :X] [RT :Y]
```

Chapter 14
Diverse functions

INTRODUCTION

Certain primitives affect the Logo system itself. You can use them to access the computer memory or its contents directly.

Because you can accidentally destroy the contents of your workspace, be sure you have saved all your work before using them. In general, the names of such primitives start with a. (dot).

NODES **operation**

Returns the number of free nodes: how much space remains in your workspace for procedures, variables, and running procedures. A node occupies five bytes of memory. The use of NODES immediately after RECYCLE will tell you how many nodes are still free.

RECYCLE **command**

Frees as many nodes as possible by performing a 'garbage collection'.
Garbage is collected automatically as the workspace becomes full, but each time it takes one second.

The use of RECYCLE prevents the automatic garbage collector from slowing things down at an inopportune time.

.CONTENTS **command**

Outputs a list of everything Logo knows. This includes your procedures and variables, and most of the things you've typed in. Note: CONTENTS can use a lot of node space.

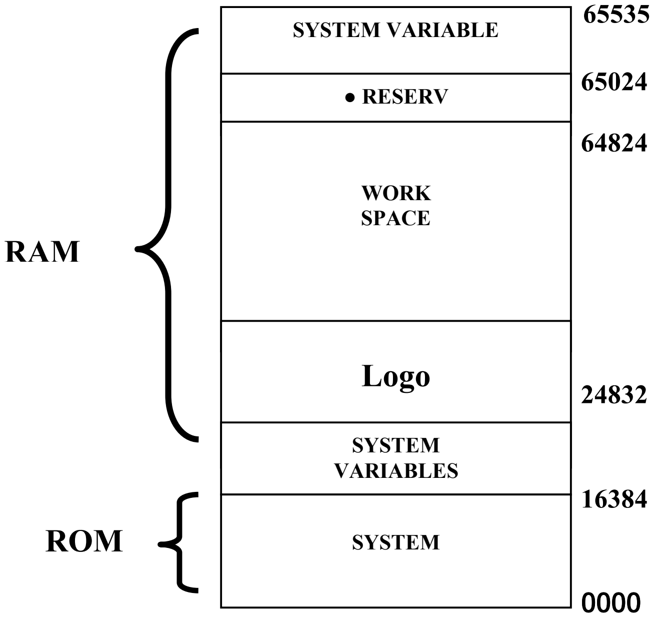
.PRIMITIVES **command**

Prints out the Logo primitives.

.RESERVE n command

If you wish to load a machine code program, you may reserve a place for it in the Logo workspace, specifying how many bytes you wish to reserve. .RESERVE n tells Logo to reserve n bytes for holding a machine code program. .RESERVE n can be used ONLY at the beginning of a Logo session.

For example, suppose you wish to reserve 200 bytes of memory to load a machine code program You give the command .RESERVE 200. The diagram below illustrates what will happen in Logo's memory.



.RESERVED command

Returns the beginning and end addresses of the area reserved by .RESERVE.

.BLOAD "filename address command

Loads a file from your cassette or Microdrive into memory at the given address. In this example, the start address is 64824.

```
.BLOAD "PICTURE 64824
```

BSAVE "filename [start address length] command

Looks for the address given and saves n bytes under a file called filename. In this example, the start address is 64824 and the size is 200.

```
.BSAVE "MAP C64824 2003
```

.SETSERIAL n

command

Takes the baud rate n and then sets the speed of transmission. The baud rate may be: 50,110, 300, 600,1200,2400,4800,9600,or 19200.
The default for the baud rate is 9600.

SERIALIN

command

Reads everything that arrives at the serial port (RS232 Interface) at the set baud rate, and outputs a byte between 0 and 255.

.SERIALOUT n

command

Sends a byte to the serial port (RS232 Interface).

.DEPOSIT address n

command

Places the value n in the location specified by the address.

```
.DEPOSIT 65010 10
```

The following example shows how .DEPOSIT can be used to make a user defined character which can later be printed.

```
TO USR.A
DEPOSIT 65368 64
DEPOSIT 65369 68
DEPOSIT 65370 72
DEPOSIT 65371 80
DEPOSIT 65372 42
DEPOSIT 65373 74
DEPOSIT 65374 15
DEPOSIT 65375 2
PR CHAR 144
END

USR.A
¼

TO DEFCHAR :L
IF EMPTYP :L [STOP]
.DEPOSIT 65368 + COUNT :LIST LAST
 :L
DEFCHAR BF :L
END
```

Remember there are twenty-one user-definable graphics characters which can be printed using PR CHAR 144...164.

.EXAMINE address **command**

Recovers a value stored at the specified address.

```
PRINT .EXAMINE 65010
10
```

.CALL address **command**

Runs a machine code program previously installed by a. BLOAD

Appendix 1
Logo messages

Not enough inputs to ...
I don't know how to ...
You don't say what to do with ...
... does not output to ...
... is used by Logo
... is already defined
... is not true or false
... is not a word
... defined
Too many inside parentheses
... open.file problem
... file not found
Bad file name
You're at toplevel
STOPPED!!!
Turtle out of field
Not enough space to proceed
... doesn't like
... has no value
... is a primitive
Not enough items in ...
Overflow
... can't divide by zero
... number too big
... as input
... in

Appendix 2
ASCII character set

Code	Character	Code	Character
		38	&
0	not used	39	‘
1		40	(
2		41	)
3		42	*
4		43	+
5		44	,
6	PRINT comma	45	_
7	EDIT	46	.
8	cursor left	47	/
9	cursor right	48	0
10	cursor down	49	1
11	cursor up	50	2
12	DELETE	51	3
13	ENTER	52	4
14	number	53	5
15	not used	54	6
16	INK control	55	7
17	PAPER control	56	8
18	FLASH control	57	9
19	BRIGHT control	58	:
20	INVERSE control	59	;
21	OVER control	60	<
22	AT control	61	=
23	TAB control	62	>
24	not used	63	?
25		64	@
26		65	A
27		66	B
28		67	C
29		68	D
30		69	E
31		70	F
32	SPACE	71	G
33	!	72	H
34	"	73	I
35	#	74	J
36	$	75	K
37	%		

Code	Character	Code	Character	
76	L	121	y	
77	M	122	z	
78	N	123	{	
79	O	124	|	
80	P	125	}	
81	Q	126	~	
82	R	127	©	
83	S	128	□	
84	T	129	▝	
85	U	130	▘	
86	V	131	▀	
87	W	132	▗	
88	X	133	▐	
89	Y	134	▚	
90	Z	135	▜	
91	[	136	▖	
92	/	137	▞	
93	]	138	▌	
94	↑	139	▜	
95	_ (underscore)	140	▄	
96	#	141	▟	
97	a	142	▙	
98	b	143	█	
99	c	144	(a)	
100	d	145	(b)	
101	e	146	(c)	
102		147	(d)	
103	f	148	(e)	
104	g	149	(f)	
105	h	150	(g)	
106	i	151	(h)	
107	j	152	(i)	
108	k	153	(j)	
109	l	154	(k)	user
110	m	155	(l)	graphics
111	n	156	(m)	
112	o	157	(n)	
113	p	158	(o)	
114	q	159	(p)	
115	r	160	(q)	
116	s	161	(r)	
117	t	162	(s)	
117	u	163	(t)	
118	v	164	(u)	
119	w	165	RND	
120	x			

Index

(to jump to a page click on the page number)

(to jump to a page click on the page number)

(to jump to a page click on the page number)